**🎯 Updated Execution Plan (1st Release Focus)**

* **Phase A — Rapid Data Ingestion (Admin-side Web Crawl) — Start now**

Goal: Admin pastes a URL; system scrolls the page, collects info, stores it as “extracted” records; admin reviews/edits then approves → canonical DB.

**A1. Backend: Crawl pipeline (MVP)**

* **Tech**: Python **Playwright** (handles heavy JS + infinite scroll), **FastAPI** endpoint + **Celery** (or simple BackgroundTasks) with **Redis**.
* **Why Playwright**: Reliable headless browser; easy auto-scroll, wait-for-network-idle.

**New packages**

pip install playwright celery redis pydantic[email] trafilatura python-slugify

playwright install

**DB tables (new)**

* source\_page (id, url, site, status: queued|success|error, http\_status, fetched\_at)
* scrape\_job (id, url, created\_by\_admin\_id, status, error, started\_at, finished\_at)
* extracted\_party (id, page\_id, name, abbrev, logo\_url, description, raw\_json, moderation\_status: pending|approved|rejected)
* extracted\_candidate (id, page\_id, full\_name, party\_guess, constituency\_guess, photo\_url, bio, raw\_json, moderation\_status)
* **Existing canonical**: party, candidate, constituency, election (if you don’t have constituency & election, add minimal tables now).

**Endpoints (admin)**

* POST /api/v1/ingest/jobs → create crawl job from a URL
* GET /api/v1/ingest/jobs → list jobs (filter by status)
* GET /api/v1/ingest/jobs/{id} → job details + extracted records
* POST /api/v1/ingest/extracted/{type}/{id}/approve → promote extracted → canonical
* POST /api/v1/ingest/extracted/{type}/{id}/reject → mark rejected
* POST /api/v1/ingest/extracted/{type}/{id}/update → admin edits before approval

**Playwright crawler (sketch)**

# app/services/crawler.py

from playwright.async\_api import async\_playwright

import asyncio, json

async def crawl\_and\_extract(url: str) -> dict:

async with async\_playwright() as pw:

browser = await pw.chromium.launch(headless=True)

page = await browser.new\_page()

await page.goto(url, wait\_until="domcontentloaded")

# naive auto-scroll

for \_ in range(12):

await page.mouse.wheel(0, 12000)

await asyncio.sleep(0.8)

# Option A: generic extraction (full text + images)

html = await page.content()

# Option B: site-specific CSS selectors if known

# e.g., cards = await page.query\_selector\_all(".candidate-card")

# parse text content / attributes

await browser.close()

# Return both raw HTML and any parsed fields; store raw\_json for moderation

return {"html": html, "entities": []} # fill with parsed parties/candidates

**Worker**

* **MVP**: BackgroundTasks to run crawl\_and\_extract() and insert extracted\_\*.
* **Next**: switch to **Celery + Redis** for robust queue/retries.

**Moderation (promotion)**

* Convert an extracted\_party → party row (upsert by slug/name).
* Convert an extracted\_candidate → candidate (link to party if found/created).
* Keep the source\_page\_id for provenance.

**A2. Admin Panel (React): Crawl UI**

**New pages/components**

* /ingest → “Add Source URL” form + table of jobs (status chips: queued / running / success / error).
* /ingest/{jobId} → show extracted parties/candidates side-by-side editor:
  + left: extracted fields
  + right: editable form (name, party, constituency, photo URL, bio)
  + actions: **Approve**, **Reject**, **Save changes**

**API client calls (examples)**

// src/services/ingest.ts

import api from "../lib/axios";

export const createIngestJob = (url: string) =>

api.post("/ingest/jobs", { url }).then(r => r.data);

export const listIngestJobs = () =>

api.get("/ingest/jobs").then(r => r.data);

export const getIngestJob = (id: string) =>

api.get(`/ingest/jobs/${id}`).then(r => r.data);

export const approveExtracted = (type: "party"|"candidate", id: string, payload?: any) =>

api.post(`/ingest/extracted/${type}/${id}/approve`, payload).then(r => r.data);

**Notes / guardrails**

* Respect site **Terms of Service** and **robots.txt** where required.
* Add rate-limits on POST /ingest/jobs.
* Persist the **raw HTML** and a **screenshot** (Playwright page.screenshot) to help admins validate.
* **Phase B — Android Guest Mode (no login)**

Goal: Anyone can **browse election info** (elections → constituencies → parties → candidates), search, and view details.

**B1. Backend: Public read endpoints (no auth)**

* GET /api/v1/public/elections
* GET /api/v1/public/elections/{id}/constituencies
* GET /api/v1/public/parties?search=&page=&size=
* GET /api/v1/public/candidates?constituency=&party=&q=&page=&size=
* GET /api/v1/public/candidates/{id}
* **Add caching** (simple in-process or Redis) for hot lists.

**Return shapes**

* Consistent list response: { items: [...], page: 1, size: 20, total: 432 }
* Candidate detail includes joined fields (party name, constituency name).

**B2. Android app: UI (Retrofit, no auth)**

**Screens**

* Home (tabs): **Elections**, **Parties**, **Candidates**, **Search**
* List + Detail screens
* Basic filters: party / constituency
* Optional: simple caching in memory; Room can wait until R2.

**Retrofit API**

// ApiService.kt

@GET("api/v1/public/parties")

suspend fun getParties(@Query("search") q: String? = null,

@Query("page") page: Int = 1,

@Query("size") size: Int = 20): Response<Paged<Party>>

@GET("api/v1/public/candidates")

suspend fun getCandidates(@Query("party") party: String? = null,

@Query("constituency") c: String? = null,

@Query("q") q: String? = null,

@Query("page") page: Int = 1,

@Query("size") size: Int = 20): Response<Paged<Candidate>>

**Design**

* Keep it performant; paginate; show skeleton loaders.
* No login gate anywhere in guest mode.
* **Phase C — “Good enough now, better later”**

You said we can be liberal about authenticity/KYC in v1—that’s fine. Add **flags** so we can harden later without breaking clients:

* Tables: add verified\_status (unknown|verified|disputed).
* API: include verified\_status and source\_count in list/detail.
* Android & Web: show a small badge (“Unverified”)—non-blocking.

**📦 Backlog Items (to schedule right after MVP)**

* **Async SQLAlchemy** migration (convert sync endpoints to async).
* **Validation** (BD mobile format, optional email format).
* **Better extraction**: site-specific selector packs, heuristics (Trafilatura), or ML NER later.
* **Full queue**: switch BackgroundTasks→Celery; retries; visibility timeouts.
* **Image handling**: download and host candidate/party logos locally (S3/minio).
* **Search**: SQLite FTS / Postgres tsvector / Typesense for fast searching.

**🧭 Day-by-Day “Do This Next” (1 week)**

**Day 1–2 (Admin Crawl MVP)**

* [BE] Create tables & Alembic migration for: source\_page, scrape\_job, extracted\_party, extracted\_candidate.
* [BE] Implement POST /ingest/jobs (enqueue background crawl).
* [BE] Implement simple Playwright crawler (auto-scroll + content grab).
* [BE] Store raw HTML + attempt basic extraction (name, party guess).
* [FE] Add /ingest page with URL form + job list.

**Day 3–4 (Moderation)**

* [BE] GET /ingest/jobs/{id} returns extracted records.
* [FE] Review UI to edit/approve/reject; hit /approve endpoint to upsert canonical party/candidate.
* [BE] Link candidate.party\_id on approval (create party if missing).

**Day 5–6 (Android Guest Mode)**

* [BE] Implement public read endpoints (elections, parties, candidates, details) + pagination.
* [AN] Add guest tabs + lists + detail screens; hook to public endpoints.
* [AN] Add simple search + filters (party/constituency).

**Day 7 (Polish & Release)**

* [BE] Add simple cache on public endpoints.
* [ALL] Smoke test: crawl → review → approve → visible in Android guest lists.
* Tag v1.

**🧱 Minimal schemas to add (sketch)**

-- source pages

CREATE TABLE source\_page (

id UUID PRIMARY KEY,

url TEXT UNIQUE NOT NULL,

site TEXT,

status TEXT NOT NULL DEFAULT 'queued',

http\_status INT,

fetched\_at TIMESTAMPTZ,

screenshot\_path TEXT,

created\_at TIMESTAMPTZ DEFAULT now()

);

CREATE TABLE scrape\_job (

id UUID PRIMARY KEY,

url TEXT NOT NULL,

created\_by\_admin UUID REFERENCES admin(id),

status TEXT NOT NULL DEFAULT 'queued',

error TEXT,

started\_at TIMESTAMPTZ,

finished\_at TIMESTAMPTZ,

created\_at TIMESTAMPTZ DEFAULT now()

);

CREATE TABLE extracted\_party (

id UUID PRIMARY KEY,

page\_id UUID REFERENCES source\_page(id),

name TEXT,

abbrev TEXT,

logo\_url TEXT,

description TEXT,

raw\_json JSONB,

moderation\_status TEXT NOT NULL DEFAULT 'pending',

created\_at TIMESTAMPTZ DEFAULT now()

);

CREATE TABLE extracted\_candidate (

id UUID PRIMARY KEY,

page\_id UUID REFERENCES source\_page(id),

full\_name TEXT,

party\_guess TEXT,

constituency\_guess TEXT,

photo\_url TEXT,

bio TEXT,

raw\_json JSONB,

moderation\_status TEXT NOT NULL DEFAULT 'pending',

created\_at TIMESTAMPTZ DEFAULT now()

);

We’ll add/adjust canonical tables as needed (party, candidate, constituency, election).

**✅ Quick tasks you can do immediately**

1. **Create feature branches**

# backend

git checkout -b feat/ingest-crawl-mvp

# web admin

git checkout -b feat/admin-ingest-ui

# android

git checkout -b feat/guest-mode

1. **Backend**
   * Add tables + Alembic migration for ingest/extracted.
   * Create app/services/crawler.py and app/api/v1/endpoints/ingest.py with endpoints above.
   * Wire router in app/api/v1/api.py:  
     api\_router.include\_router(ingest.router, tags=["ingest"])
2. **Web admin**
   * New route /ingest: URL form → POST /ingest/jobs; job list GET /ingest/jobs.
   * Job details page: display extracted records with approve/reject.
3. **Android**
   * Add Retrofit calls to /api/v1/public/\*.
   * Build guest lists → details.

If you want, I can generate **starter code** for:

* app/api/v1/endpoints/ingest.py (FastAPI endpoints)
* app/services/crawler.py (Playwright auto-scroller + stub extractor)
* React **/ingest** page scaffold
* Android ApiService public endpoints + simple list screen

Just say “generate the backend ingest endpoints” (or the UI/Android stubs), and I’ll drop in complete files ready to paste.